



Testing for Software Safety

OSMA Software Assurance Symposium 2007

Ken Chen, JSC

Yann-Hang Lee, ASU

W. Eric Wong, UT-Dallas

Dianxiang Xu, North Dakota State University

September, 2007

Objective

- This research focuses on testing whether or not the hazardous conditions identified by design-level fault tree analysis will occur in the target implementation.
 - Part 1: Integrate fault tree models into functional specifications so as to identify testable interactions between intended behaviors and hazardous conditions.
 - Part 2: Develop a test generator that produces not only functional tests but also safety tests for a target implementation in a cost-effective way
 - Part 3: Develop a testing environment for executing generated functional and safety tests and evaluating test results against expected behaviors or hazardous conditions. It includes a test harness as well as an environment simulation of external events and conditions.

Current Work

- Goal
 - Integration of results from hazard analysis in fault trees with functional specifications in UML behavior state machines
- Challenges
 - Identify testable interactions between intended behaviors and hazardous conditions
 - Resolve the mismatch between fault tree models and functional specifications
 - Some events or unsafe states in a fault tree model may not be found or may have no relevant parts in the corresponding functional specifications



UML Behavioral State Machines

- A UML behavioral state machine can be used to specify the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.
- The behavioral state machine formalism described in UML is an object-based variant of Harel statecharts

Fault Tree Analysis (FTA)





- Useful for reliability and safety analysis
 - First used by Bell Telephone Laboratories in connection with the safety analysis of the Minuteman missile launch control system in 1962
- A top-down approach starting with *an undesirable event* called *a top event* and then determining all the ways it can happen
 - Identify all the top events to be analyzed
 - Identify the events that directly contribute to the top level vent
 - Continue this process until the lowest level defined or basic level is reached
- Important because if there is a critical failure mode, then all possible ways that mode could occur must be discovered

Fault Tree (FT)


- A fault tree is a graphical model of various parallel and sequential combinations of *faults* that will result in the occurrence of the *predefined undesired event*
 - The *undesired event* constitutes the *top event* in a fault tree constructed for the system, and generally consists of a complete, or catastrophic failure
 - The *faults* can be events that are associated with component hardware failures, human errors, or any other pertinent events which can lead to the top event
- A fault tree is composed of a number of “event” symbols and “gate” symbols
 - An *event* symbol serves to represent an initiating fault event, an event that is normally expected to occur, a condition or restriction, or a fault event which occurs because of one or more antecedent causes acting through logic gates
 - A *gate* serves to permit or inhibit the passage of fault logic up the tree, and shows the relationships of events needed for the occurrence of a *higher* event
 - The *higher* event is the *output* of the gate
 - The *lower* events are the *inputs* to the gate

Events of Fault Tree






- Primary Event

- Basic event: An event does not require any further development in order to initiate a fault
 - Fault tree symbol: 
- Conditioning event: An event describes specific conditions or restrictions that apply to any logic gate
 - Fault tree symbol: 
 - Used primarily with PRIORITY AND and INHIBIT gates
- Undeveloped event: An event which is not further developed because it is of insufficient consequence or because information is not available
 - Fault tree symbol: 
- External event: An event which is normally expected to occur
 - Fault tree symbol: 

- Intermediate Event

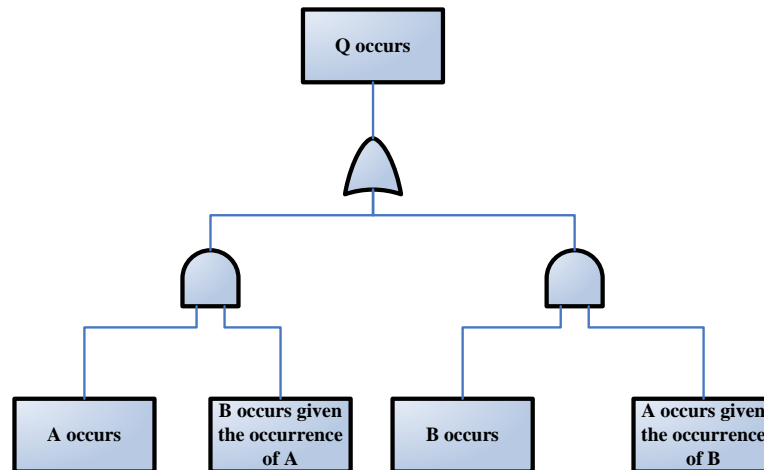
- An event that occurs because of one or more antecedent causes acting through logic gates
 - Fault tree symbol: 

Gates of Fault Tree

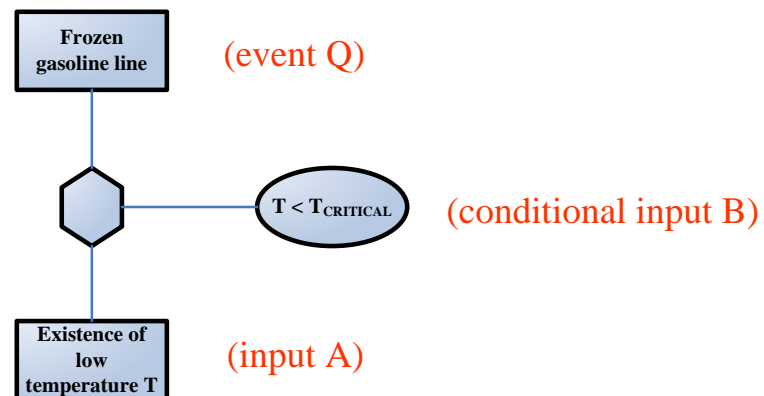
- AND-Gate
 - Output fault occurs if all of the input faults occur
 - Fault tree symbol: 
- OR-Gate
 - Output fault occurs if at least one of the input faults occurs
 - Fault tree symbol: 
- EXCLUSIVE OR-Gate
 - Output fault occurs if exactly one of the input faults occurs
 - Fault tree symbol: 
- PRIORITY AND-Gate
 - Output fault occurs if all of the input faults occur in a specific sequence (the sequence is represented by a CONDITIONING EVENT drawn to the right of the gate)
 - Fault tree symbol: 
- INHIBIT-Gate
 - Output fault occurs if the (single) input fault occurs in the presence of an enabling condition (the enabling condition is represented by a CONDITIONING EVENT drawn to the right of the gate)
 - Fault tree symbol: 

Examples of Gates (1)

- AND-Gate relationship with dependency explicitly shown

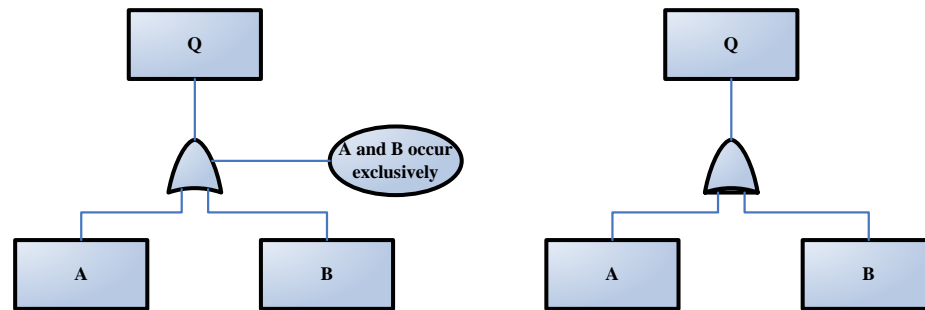


- Inhibit-Gate: event Q occurs only if input A occurs under the condition specified by input B

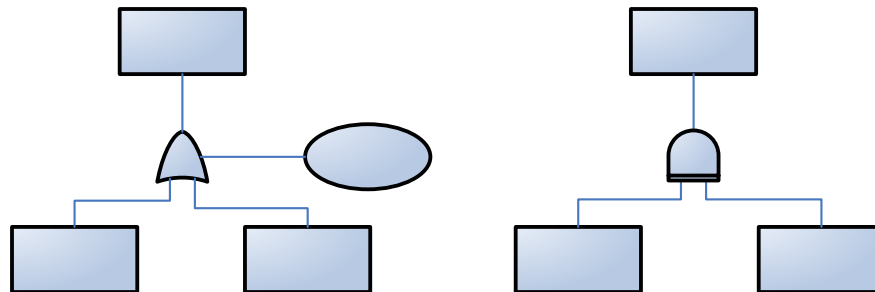


Examples of Gates (2)

- The INHIBIT-Gate is a special case of the AND-Gate. The output is caused by a single input, but some qualifying condition must be satisfied before the input can produce the output
- The EXCLUSIVE OR-Gate is a special case of the OR-Gate in which the output event occurs only if exactly one of the input events occurs



- The PRIORITY AND-Gate is a special case of the AND-Gate in which the output event occurs if all input events occur in a specified ordered sequence



Construction of Fault Tree

- A fault tree can be constructed based on the Failure Modes and Effects Analysis (FMEA) and **system block diagrams**
- Rules of thumb
 - **No Miracles Rule**: If the normal functioning of a component propagates a fault sequence, then it is assumed that the component functions normally.
 - **Complete-the-Gate Rule**: All inputs to a particular gate should be completely defined before further analysis of any of them is undertaken.
 - **No Gate-to-Gate Rule**: Gate inputs should be properly defined fault events, and the gates should not be directly connected to other gates.

Boolean Algebra & Fault Tree Analysis

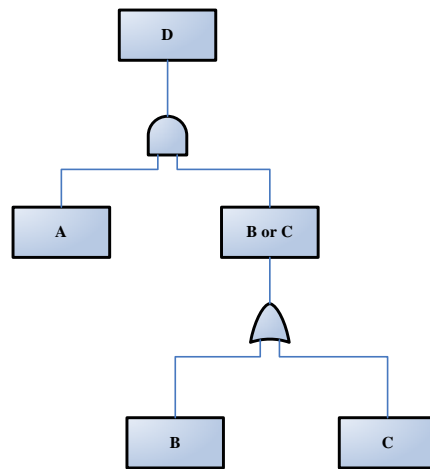
- The OR-Gate is equivalent to the Boolean symbol “+”
- The AND-Gate is equivalent to the Boolean symbol “•”
- Rules of Boolean Algebra

	Mathematical Symbolism	Engineering Symbolism	Designation
(1a)	$X \cap Y = Y \cap X$	$X \cdot Y = Y \cdot X$	Commutative Law
(1b)	$X \cup Y = Y \cup X$	$X + Y = Y + X$	
(2a)	$X \cap (Y \cap Z) = (X \cap Y) \cap Z$	$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ $X(YZ) = (XY)Z$	Associative Law
(2b)	$X \cup (Y \cup Z) = (X \cup Y) \cup Z$	$X + (Y + Z) = (X + Y) + Z$	
(3a)	$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$	$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$ $X(Y + Z) = XY + XZ$	Distributive Law
(3b)	$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$	$X + Y \cdot Z = (X + Y) \cdot (X + Z)$	
(4a)	$X \cap X = X$	$X \cdot X = X$	Idempotent Law
(4b)	$X \cup X = X$	$X + X = X$	
(5a)	$X \cap (X \cup Y) = X$	$X \cdot (X + Y) = X$	Law of Absorption
(5b)	$X \cup (X \cap Y) = X$	$X + X \cdot Y = X$	
(6a)	$X \cap X' = \phi$	$X \cdot X' = \phi$	Complementation
(6b)	$X \cup X' = \Omega = I^*$	$X + X' = \Omega = I$	
(6c)	$(X')' = X$	$(X')' = X$	
(7a)	$(X \cap Y)' = X' \cup Y'$	$(X \cdot Y)' = X' + Y'$	de Morgan's Theorem
(7b)	$(X \cup Y)' = X' \cap Y'$	$(X + Y)' = X' \cdot Y'$	
(8a)	$\phi \cap X = \phi$	$\phi \cdot X = \phi$	Operations with ϕ and Ω
(8b)	$\phi \cup X = X$	$\phi + X = X$	
(8c)	$\Omega \cap X = X$	$\Omega \cdot X = X$	
(8d)	$\Omega \cup X = \Omega$	$\Omega + X = \Omega$	
(8e)	$\phi' = \Omega$	$\phi' = \Omega$	
(8f)	$\Omega' = \phi$	$\Omega' = \phi$	
(9a)	$X \cup (X' \cap Y) = X \cup Y$	$X + X' \cdot Y = X + Y$	These relationships are unnamed but are frequently useful in the reduction process.
(9b)	$X' \cap (X \cup Y') = X' \cap Y' = (X \cup Y)'$	$X' \cdot (X + Y') = X' \cdot Y' = (X + Y)'$	

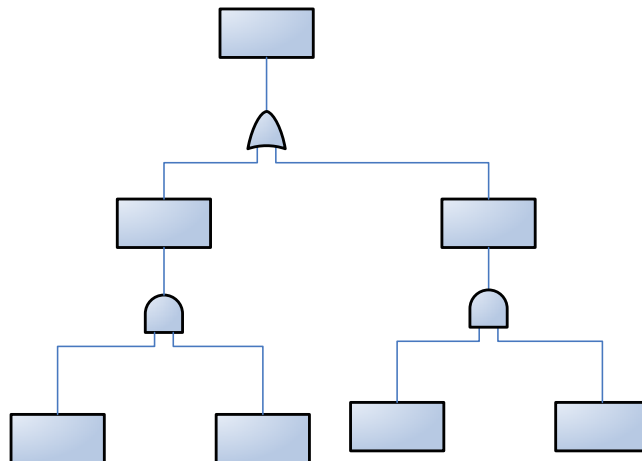
* The symbol **I** is often used instead of Ω to designate the Universal Set. In engineering notation Ω is often replaced by 1 and ϕ by 0.

Examples of Fault Trees

- Fault tree structure for $D = A \cdot (B + C)$



- An equivalent fault tree for $D = A \cdot (B + C)$



Minimal Cut Set (1)

- A *cut set* in a fault tree is *a set of basic events* whose simultaneous occurrence ensures that the top event occurs.
- A cut set is said to be *minimal* if the set cannot be reduced without losing its status as a cut set.
 - The combination is “smallest” in that all the events are needed for the top event to occur. If one of the events in the cut set does not occur, then the top event will not occur (by this combination).
- A fault tree may consist of *a finite number of minimal cut sets*, which are unique for that top event.

Minimal Cut Set (2)

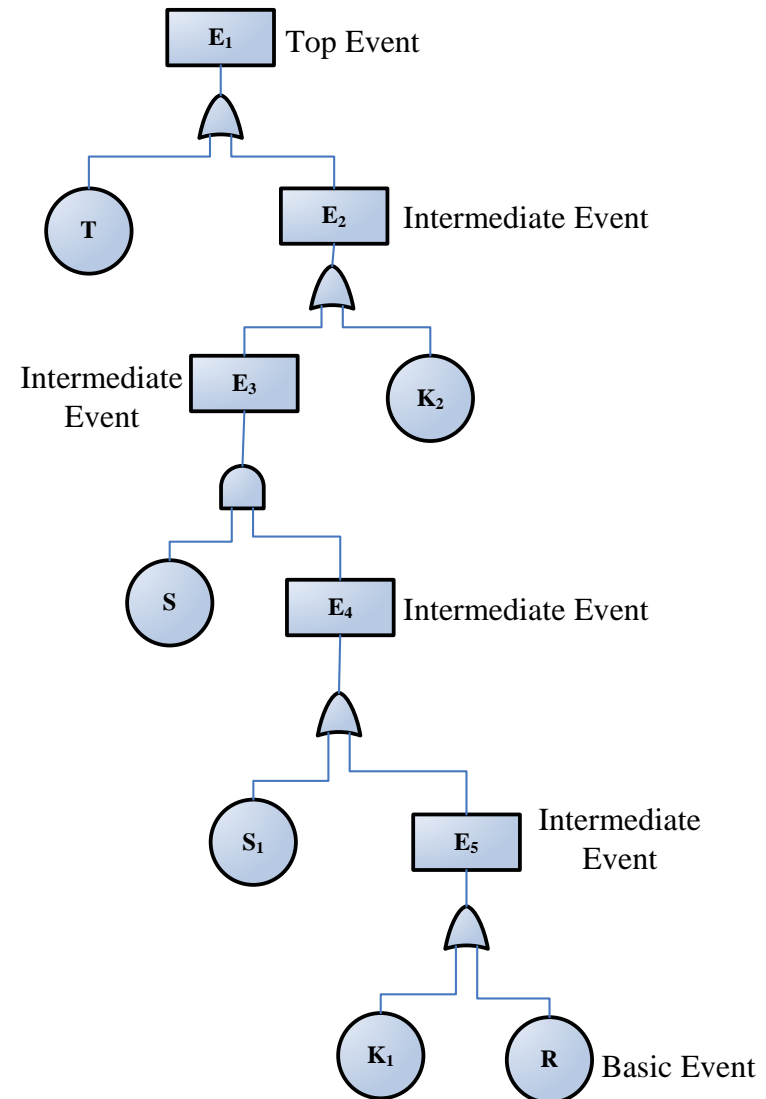
- The *one-event* minimal cut sets represent those single events which will cause the top event to occur
- The *two-event* minimal cut sets represent those pairs of events which together will cause the top event to occur
- Similarly, for an *n-event* minimal cut set, all n events in the cut set must occur in order for the top event to occur
- The minimal cut set for the top-event can be written in the following general form
 - $TOP = \mathcal{M}_1 + \mathcal{M}_2 + \dots + \mathcal{M}_m$ where TOP is the top event and \mathcal{M}_i ($1 \leq i \leq m$) is a minimal cut set
 - $\mathcal{M}_i = X_1 \bullet X_2 \bullet \dots \bullet X_n$ where X_k ($1 \leq k \leq n$) is a basic event in the fault tree

Examples of Minimal Cut Sets

- The top event can be expressed as a *Boolean function* of the basic events

$$\begin{aligned}
 E_1 &= T + E_2 \\
 &= T + (K_2 + E_3) \\
 &= T + K_2 + (S \cdot E_4) \\
 &= T + K_2 + (S \cdot (S_1 + E_5)) \\
 &= T + K_2 + (S \cdot S_1) + (S \cdot E_5) \\
 &= T + K_2 + (S \cdot S_1) + S \cdot (K_1 + R) \\
 &= T + K_2 + (S \cdot S_1) + S \cdot K_1 + S \cdot R
 \end{aligned}$$

- The above expression of the top event in terms of the basic events to the tree can be viewed as a *Boolean algebraic equivalent* of the tree itself.
- In this example, we have five minimal cut sets — two singles and three doubles
 - K_2
 - T
 - $S \cdot S_1$
 - $S \cdot K_1$
 - $S \cdot R$



Construction of Minimal Cut Set

- The tree is first translated to its *equivalent Boolean equations* and then either the “**top-down**” or “**bottom-up**” substitution method is used.
 - Both methods involve substituting and expanding Boolean expressions.
 - Two Boolean laws, the **distributive** law and the law of **absorption**, are used to remove the redundancies.
[\[Reference 3\]](#)
- Tools are available for computing the minimal cut sets of a fault tree

Minimal Cut Set Transition

- The occurrence of basic events in a minimal cut set \mathcal{M} will lead to the occurrence of an undesired event and finally cause a transition from a normal state to a fault state. We define such a transition as a **minimal cut set transition**.
- Given a fault tree \mathcal{T} and a minimal cut set \mathcal{M} , we can obtain a subtree, whose root is the top event of the fault tree, and whose leaves are the basic events in \mathcal{M} and some other primary events (e.g., external events) that cause the top event.
 - The subtree contains
 - all the basic events in \mathcal{M} (the given minimal cut set)
 - all the other necessary “occurring” primary events (undeveloped, external, and conditioning events)
 - all the necessary “occurring” intermediate events
 - the top event
- A subtree therefore describes a minimal cut set transition in the behavioral state machine

Research Issues

- How to simplify a fault tree by deleting all the infeasible and non-causal events and gates?
 - A fault tree may be constructed *independently* from the construction of the UML behavior state machine for the target system. Some constraints, which are defined implicitly or explicitly on the system or on the machine, may affect the occurrence of the primary and/or non-primary events in the fault tree. How to use these constraints to simplify the fault tree in order to reduce the complexity of the machine after combination?
 - [Proposed Solution](#)
- Given a minimal cut set of a fault tree, how to construct a corresponding subtree which covers all necessary events leading to the top event?
 - [Proposed Solution](#)
- How to transform a subtree to a minimal cut set transition?
 - [Proposed Solution](#)
- How to add the minimal cut set transitions to a UML behavior state machine?
 - [Proposed Solution](#)



Research Issue I

Simplify a fault tree by deleting all the infeasible
and non-causal events and gates

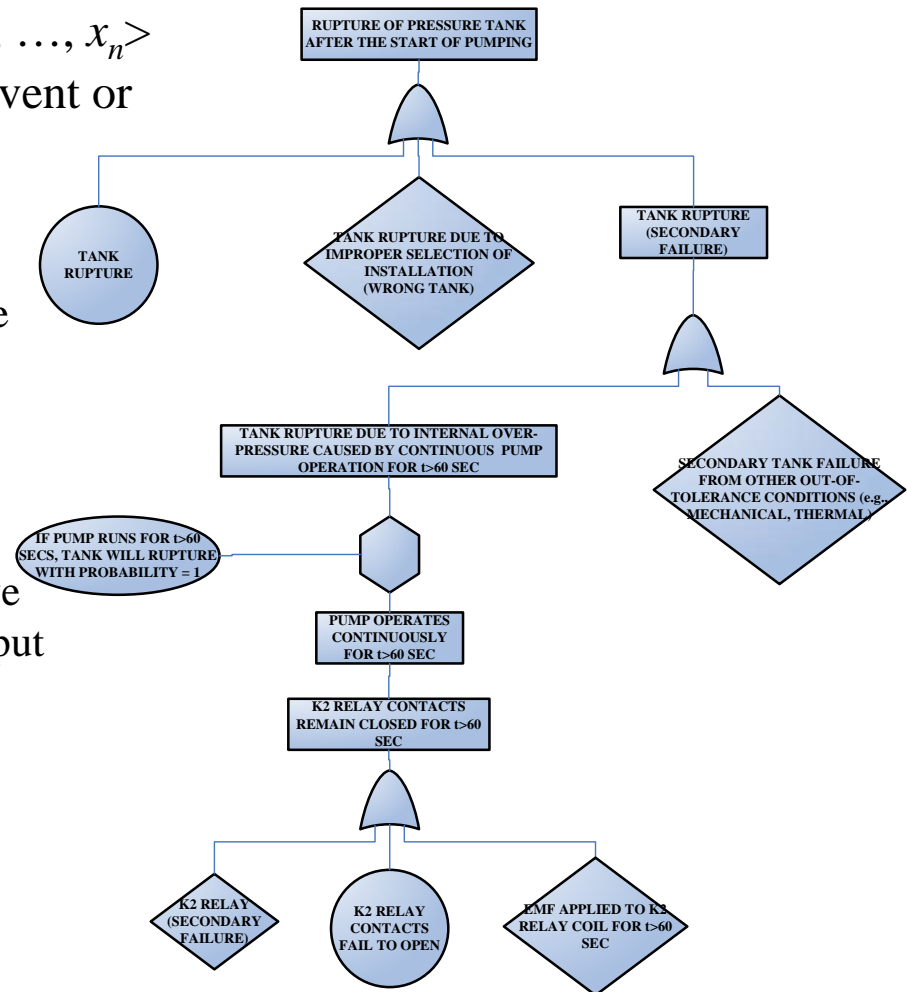
[Back to Research Issues](#)

Infeasible Minimal Cut Set (Transition)

- Theoretically, we can transform the subtrees for the minimal cut sets to minimal cut set transitions, and then extend the UML behavior state machine to describe the possibilities of the occurrence of undesired events by adding all minimal cut set transitions to the machine.
- A minimal cut set transition can be *infeasible* because of the restrictions on a specific system. These restrictions are implicitly or explicitly defined, which *make the transition impossible to traverse*. We call such a transition an “infeasible” minimal cut set transition, and the corresponding minimal cut set is an “infeasible” minimal cut set.
 - On the contrary, we have “feasible” minimal cut sets and “feasible” minimal cut set transitions.
- Identification of “infeasible” minimal cut sets beforehand can
 - reduce the complexity of the UML behavior state machine after combination
 - help the derivation of traversable test sequences and generation of effective test cases from the combined machine

Bottom-Up Path

- We define a bottom-up path $p = \langle x_1, x_2, \dots, x_n \rangle$ from x_1 to x_n , where x_i ($1 \leq i \leq n$) is an event or a gate in the fault tree
- The path needs to satisfy the following:
 - The adjacent events/gates on the path are connected by a line in the fault tree
 - If x_i is a gate, x_{i-1} is an input event of x_i or a conditioning event applied to x_i , and x_{i+1} is the output event of x_i
 - If x_i is an event, x_{i+1} can be the successive event of x_i or a gate receiving x_i as its input
- A bottom-up path gives the order of the occurrence of the events



Identification of Infeasible Events & Gates (1)

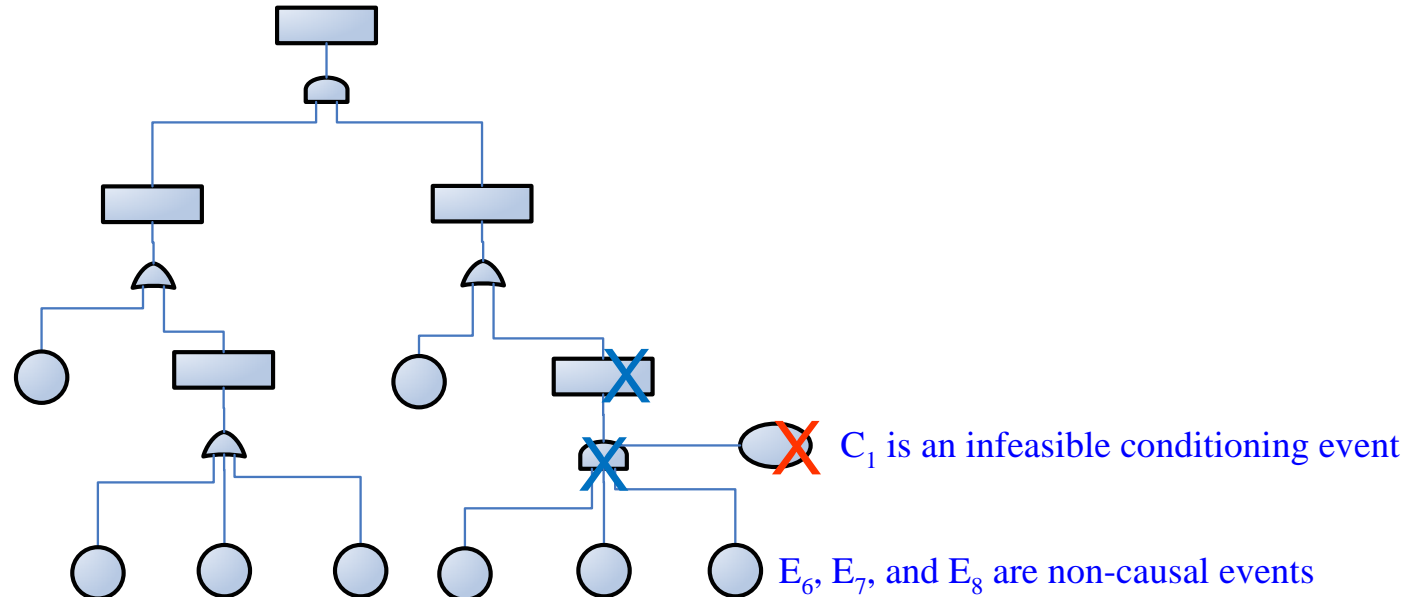
- A **primary/non-primary event** with unrealistic constraints is infeasible
 - Example: an event “a sportsman jumps for a distance of ten meters” can be regarded as an infeasible event because a human being cannot jump that far
- A **conditioning event** is infeasible if the condition specified in the event can never be reached or the probability specified in the event is 0
- A **gate** is infeasible if it cannot be passed through based on the combinations of all its input events
 - Example: an AND-Gate $I = E_1 \cdot E_2$ but events E_1 and E_2 cannot occur simultaneously because they contradict each other
 - Example: an OR-Gate $I = E_1 + E_2$ but neither E_1 nor E_2 can occur

Identification of Infeasible Events & Gates (2)

- If *any* of the input events of an **AND**-Gate is infeasible, then the AND-gate is infeasible
- If *all* the input events of an **OR**-Gate are infeasible, then the OR-Gate is infeasible
- A gate is infeasible if its conditioning event is infeasible
- The output event of an infeasible gate is infeasible
- The successive event of an infeasible event is infeasible

Identification of Non-Causal Events & Gates

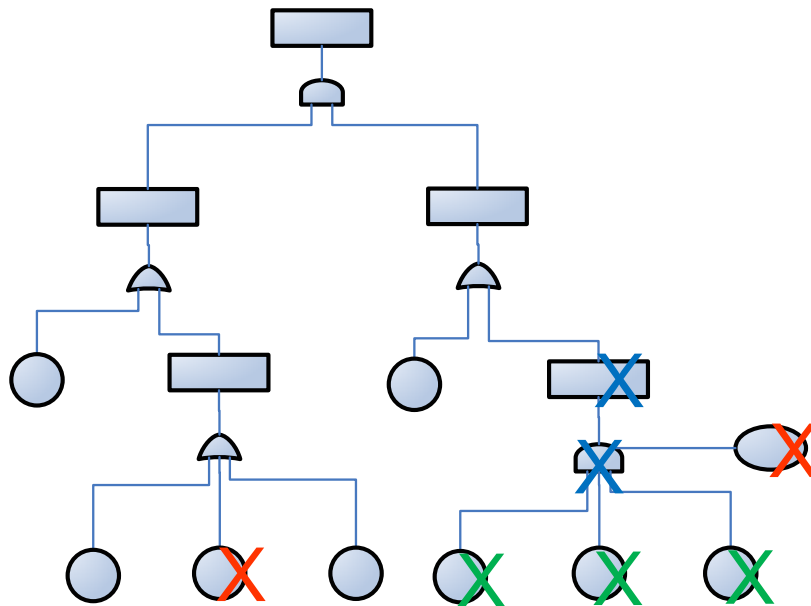
- A non-causal event/gate does not contribute to the occurrence of the top event
 - The Boolean equations for the following tree are $TOP = I_1 \cdot I_2$; $I_1 = E_1 \cdot I_3$; $I_2 = E_2 \cdot I_4$; $I_3 = E_2 \cdot E_3 \cdot E_4$; $I_4 = [C_1] (E_6 \cdot E_7 \cdot E_8)$. Suppose the conditioning event C_1 is infeasible (cannot occur), the corresponding AND-gate is infeasible which makes the intermediate event I_4 infeasible. Although the top event can still occur if E_1 and E_5 occur. Basic events E_6 , E_7 , and E_8 have no impact on the occurrence of the top event. They are defined as “non-causal” events.



- If there does not exist a bottom-up path from an event E (or a gate G) to the top event, on which all the events/gates are feasible, then E (or G) is a non-causal event (or gate) for the top event.

Simplification of a Fault Tree

- A fault tree can be simplified by removing all the infeasible and non-causal events and gates
 - In the following tree, events E_3 , I_4 , and C_1 are infeasible and events E_6 , E_7 , and E_8 are non-causal



- X: the event/gate is infeasible according to some constraints on the system
- X: the event/gate is infeasible according to some deduction rules
- X: the event/gate is a non-causal event/gate.

- If the top event is infeasible, the fault tree can be excluded from further consideration.

[Back to Research Issues](#)



Research Issue 2

Construct a subtree based on a given minimal cut set

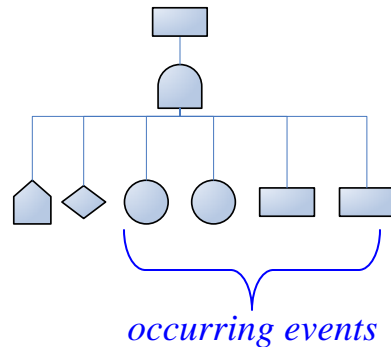
[Back to Research Issues](#)

Construction of a Subtree for a Given Minimal Cut Set (1)

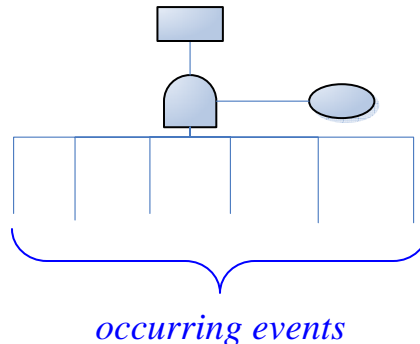
- A subtree for a minimal cut set \mathcal{M} can be obtained by simulating the occurrence of all the basic events in \mathcal{M}
 - Step 0: Assuming each event in a *simplified* fault tree can be classified as “occurring” or “non-occurring”, and each gate can be classified as “passed through” or “not-passed through”
 - Step 1: Initially, mark all the events in the fault tree as “non-occurring” and all the gates as “not-passed through”
 - Step 2: Mark a “non-occurring” basic event as “occurring” if it belongs to \mathcal{M}

Construction of a Subtree for a Given Minimal Cut Set (2)

- Step 3: Repeat steps 3.1 to 3.4 until no events in the fault tree can be marked as “occurring” and no gates can be marked as “passed through”
 - Step 3.1: Mark an external/undeveloped event E as “occurring” if E is an input of an AND-Gate G and all other basic events and intermediate events received by G are “occurring”



- Step 3.2: Mark the conditioning event C applied to a gate G as “occurring” if the “occurring” input events can satisfy the conditions indicated in C

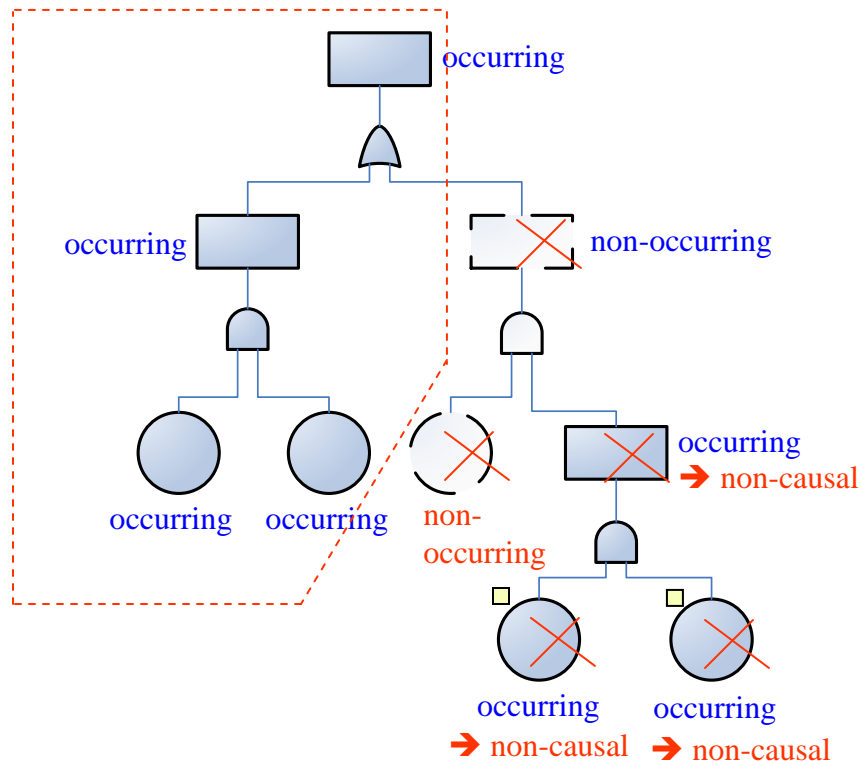


Construction of a Subtree for a Given Minimal Cut Set (3)

- ❑ Step 3.3: Mark a non-primary event E as “occurring” if E is the output of a “passed through” gate G or E is the successive event of an “occurring” event
- ❑ Step 3.4: Mark a “not-passed through” gate G as “passed through” if
 - G is an OR-Gate, and at least one of its input events is “occurring”, and the applied conditioning event, if it exists, is also “occurring”
 - G is an AND-Gate, and all of its input events are “occurring”, and the applied conditioning event, if it exists, is also “occurring”
- Step 4: Identify additional non-causal events and gates *with respect to the given minimal cut set* which are *not removed* during the simplification of the fault tree (as discussed earlier)

Construction of a Subtree for a Given Minimal Cut Set (4)

- Change an “occurring” event to “non-causal” and a “passed through” gate to “non-causal” if there does not exist a bottom-up path from the event/gate to the top event, on which all events are marked as “occurring” and all gates are marked as “passed through”



Let the minimal cut set $\mathcal{M} = \{E_1, E_2\}$

When basic events E_1 and E_2 occur, intermediate events I_1 and I_3 will occur.

Since basic event E_3 is not in \mathcal{M} ,

it will not be included in the subtree for \mathcal{M} .

That is, E_3 is regarded as “infeasible” with respect to this subtree.

As a result, I_2 cannot occur which makes E_1 and E_2 in the *right part of the tree* as “non-causal”.

It is better to represent such E_1 and E_2 as the **mirror blocks** of the E_1 and E_2 in the *left part* of the fault tree. Note that **an event may appear multiple times at different places in a fault tree and affect different parts of the tree.** [\[Reference 1\]](#)

- Step 5: Remove all “non-occurring”/“non-causal” events and all “not-passed through”/“non-causal” gates from the fault tree

Characteristics of the Subtree

- The subtree constructed at step 5 contains
 - all the basic events in \mathcal{M} (the given minimal cut set)
 - all the other necessary “occurring” primary events (undeveloped, external, and conditioning events)
 - all the necessary “occurring” intermediate events
 - the top event

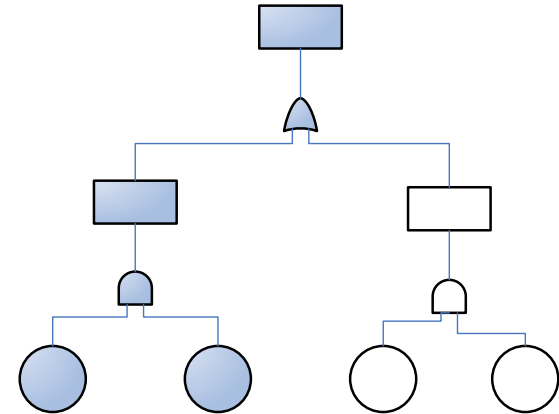
$$\text{Subtree-events } (\mathcal{M}) = \mathcal{M} \cup \{\text{other necessary “occurring” primary events}\} \cup \{\text{necessary “occurring” intermediate events}\} \cup \{\text{top event}\}$$

- The subtree for a given \mathcal{M} does not contain any basic events that are not in \mathcal{M}
- The top event of a fault tree must be included in the subtree

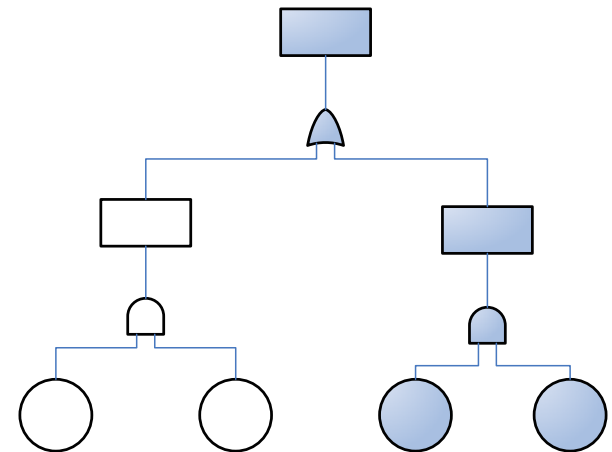
Examples of Two Subtrees

- The equivalent Boolean equations for the fault tree to the right is
 - $TOP = I_1 + I_2$
 - $I_1 = E_1 \cdot E_2$
 - $I_2 = E_3 \cdot E_4$
- Let minimal cut set $\mathcal{M}_1 = \{E_1, E_2\}$
 - Subtree-events (\mathcal{M}_1) = $\mathcal{M}_1 \cup \{I_1\} \cup \{TOP\}$
 - The equivalent Boolean equations for subtree 1
 - $T = I_1$
 - $I_1 = E_1 \cdot E_2$
- Let minimal cut set $\mathcal{M}_2 = \{E_3, E_4\}$
 - Subtree-events (\mathcal{M}_2) = $\mathcal{M}_2 \cup \{I_2\} \cup \{TOP\}$
 - The equivalent Boolean equations for subtree 2
 - $T = I_2$
 - $I_2 = E_3 \cdot E_4$

Subtree for the minimal cut set \mathcal{M}_1



Subtree for the minimal cut set \mathcal{M}_2



[Back to Research Issues](#)



Research Issue 3

Transform a subtree to a minimal cut set transition

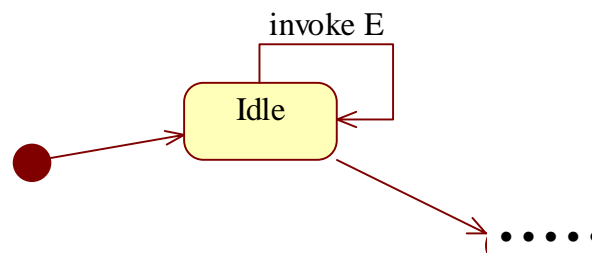
[Back to Research Issues](#)

Run-to-Completion Assumption

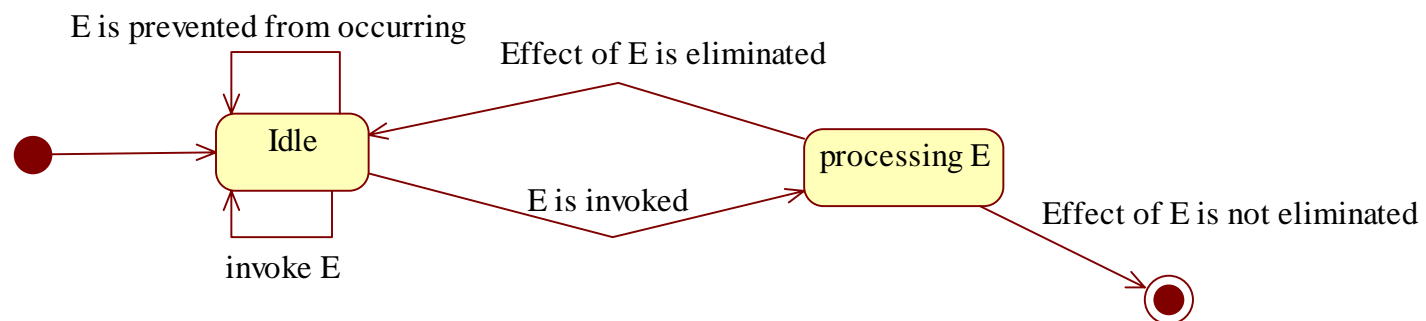
- The semantics of event occurrence processing in UML is based on the *run-to-completion* assumption, interpreted as run-to-completion processing.
 - Run-to-completion processing means that an event occurrence can only be taken from the event pool and dispatched if the processing of the previous occurrence is fully completed.
[\[Reference 2\]](#)
- Assume that the event occurrence processing in a fault tree is also based on the run-to-completion assumption.

A Primary Event in a UML Behavior State Machine (1)

- A primary event of a fault tree can be transformed to *a trigger or a guard condition for a transition* in a UML behavioral state machine.
 - The occurrence of a *basic/undeveloped/external* event in a fault tree is *semantically equivalent* to *triggering a transition* in a UML behavior state machine



- Event E may be prevented from occurring or its effect can be eliminated

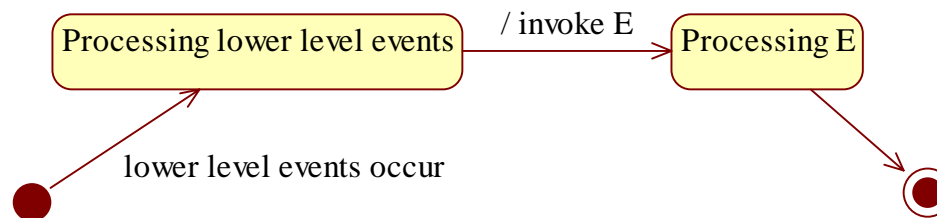


A Primary Event in a UML Behavior State Machine (2)

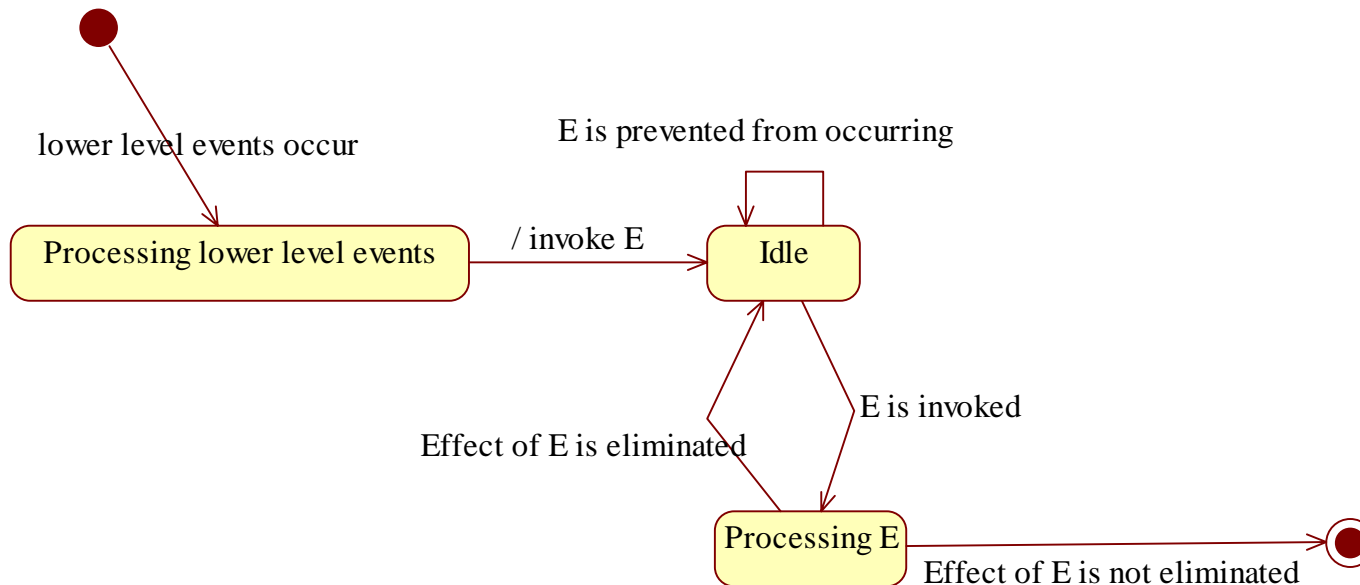
- A conditioning event can be transformed to a transition guard
 - Example:
 - Fault Tree: $T < T_CRITICAL$
 - UML behavior state machine: $[T < T_CRITICAL]$

An Intermediate Event in a UML Behavior State Machine (1)

- An intermediate event of a fault tree can be transformed to an action in a UML behavior state machine

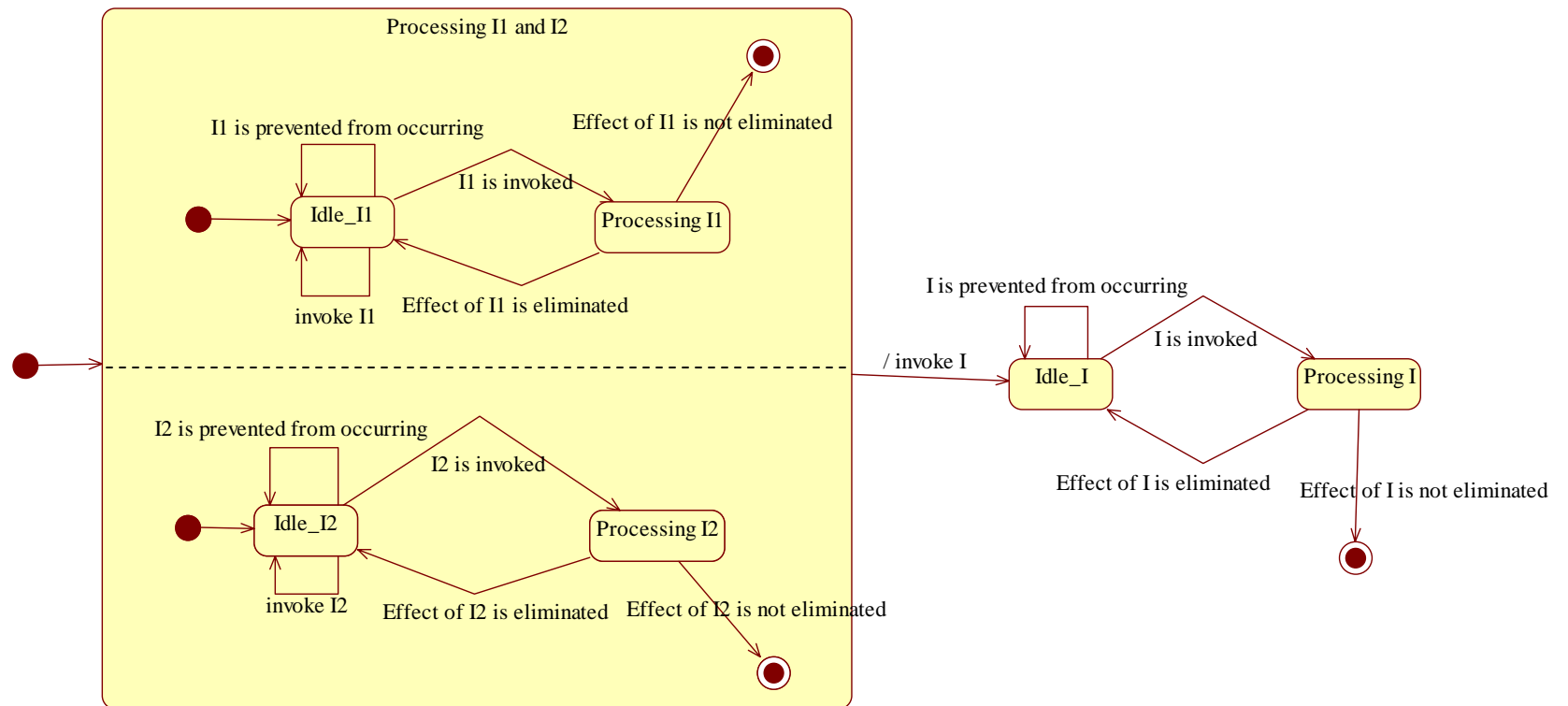


- Event E may be prevented from occurring or its effect can be eliminated



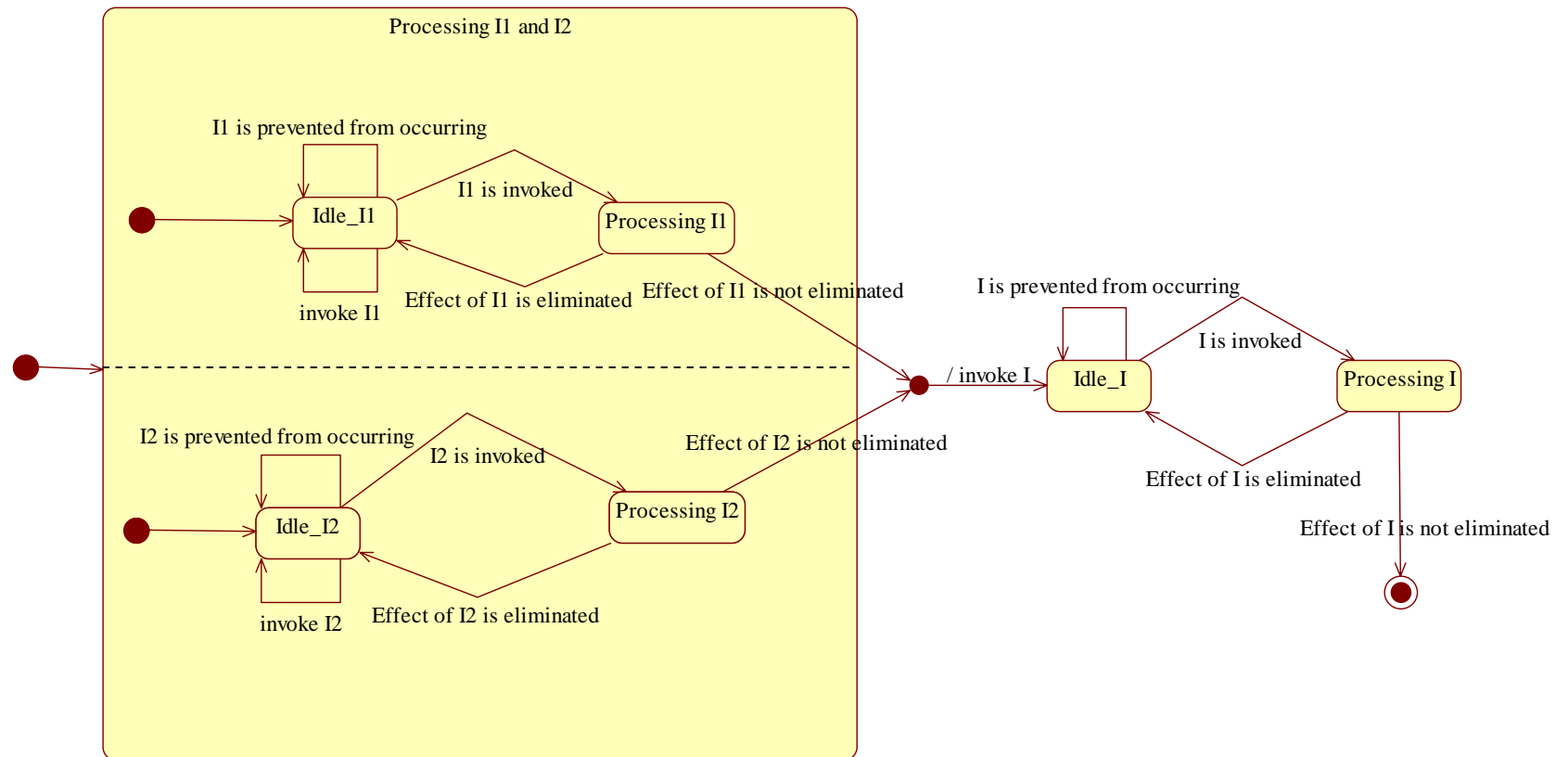
An Intermediate Event in a UML Behavior State Machine (2)

- Example: $I = I_1 \bullet I_2$



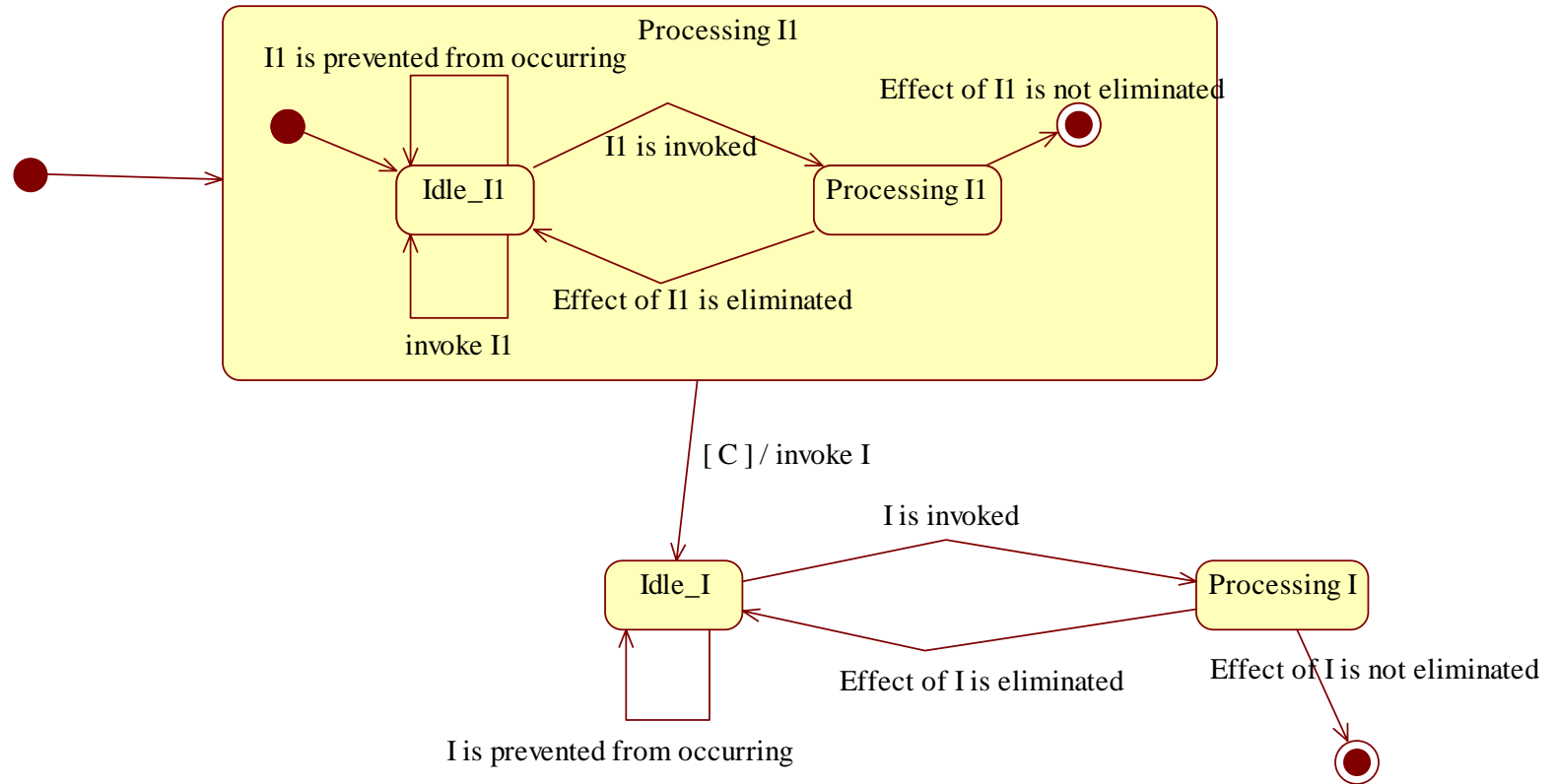
An Intermediate Event in a UML Behavior State Machine (3)

- Example 2: $I = I_1 + I_2$



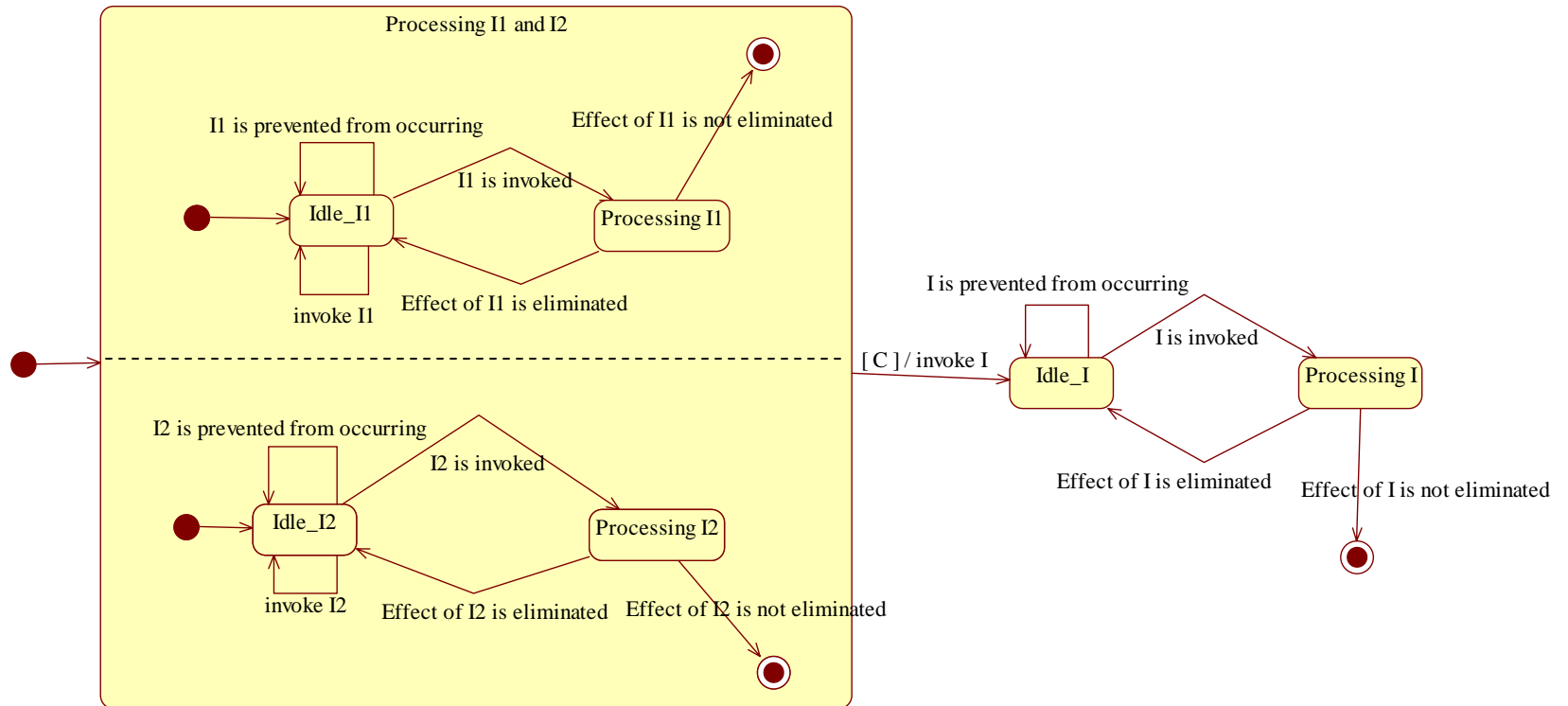
An Intermediate Event in a UML Behavior State Machine (4)

- Example 3: $I = [C] I_1$ /* if (I_1 occurs under the condition C) {I occurs} */



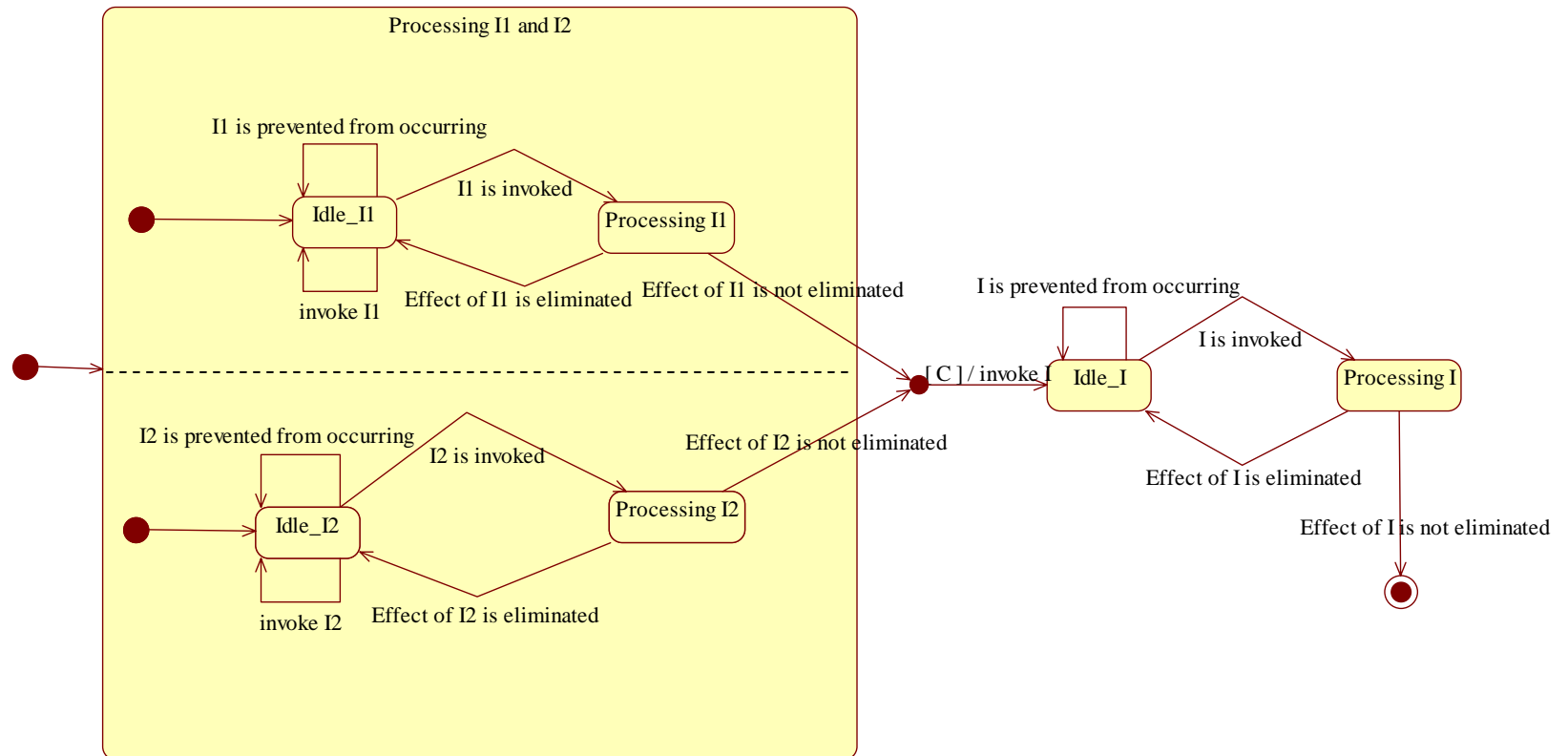
An Intermediate Event in a UML Behavior State Machine (5)

- Example 4: $I = [C] (I_1 \bullet I_2)$ /* if (I_1 and I_2 occur under the condition C) { I occurs} */



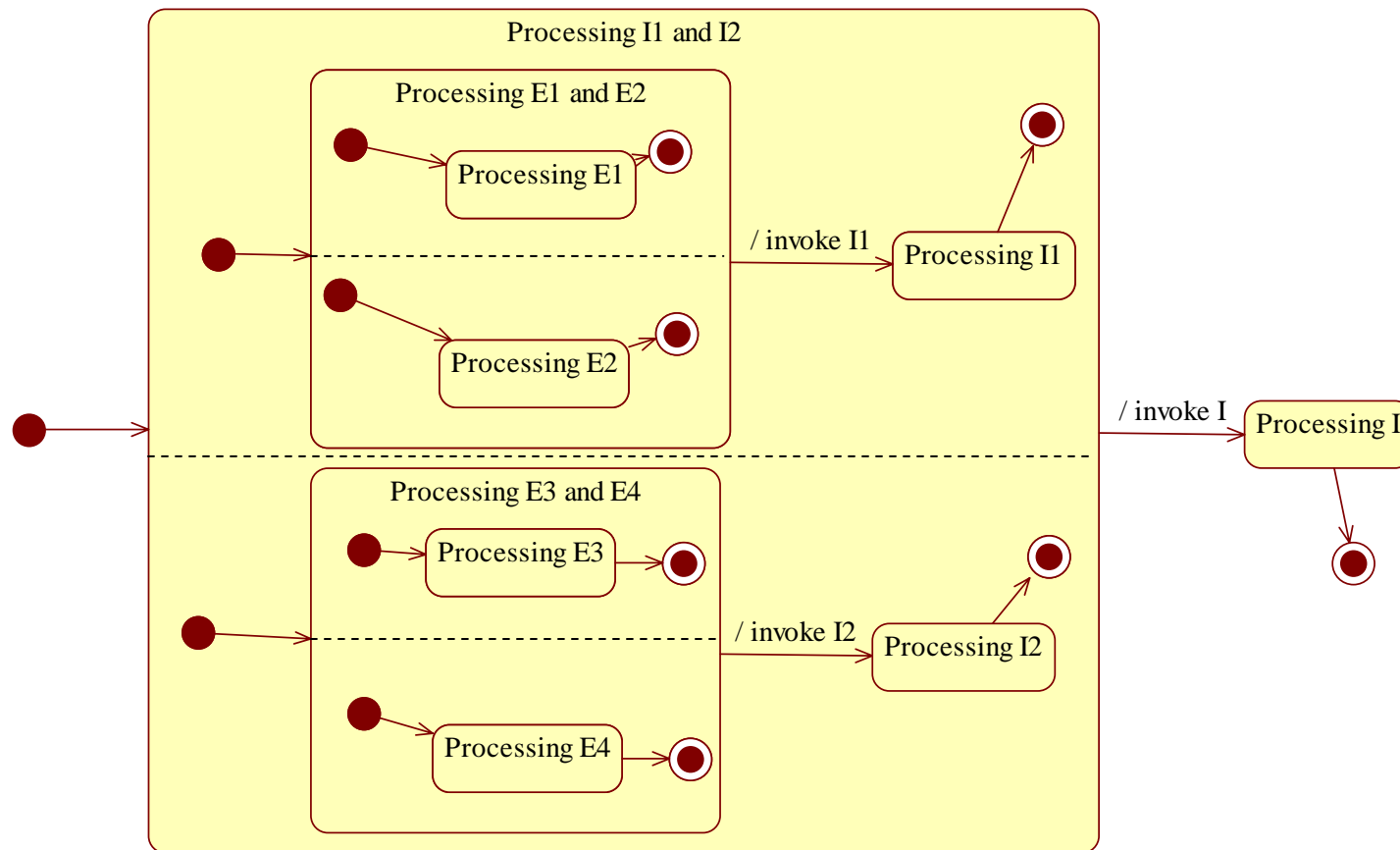
An Intermediate Event in a UML Behavior State Machine (6)

- Example 5: $I = [C] (I_1 + I_2)$ /* if (I_1 or I_2 occur under the condition C) {I occurs} */

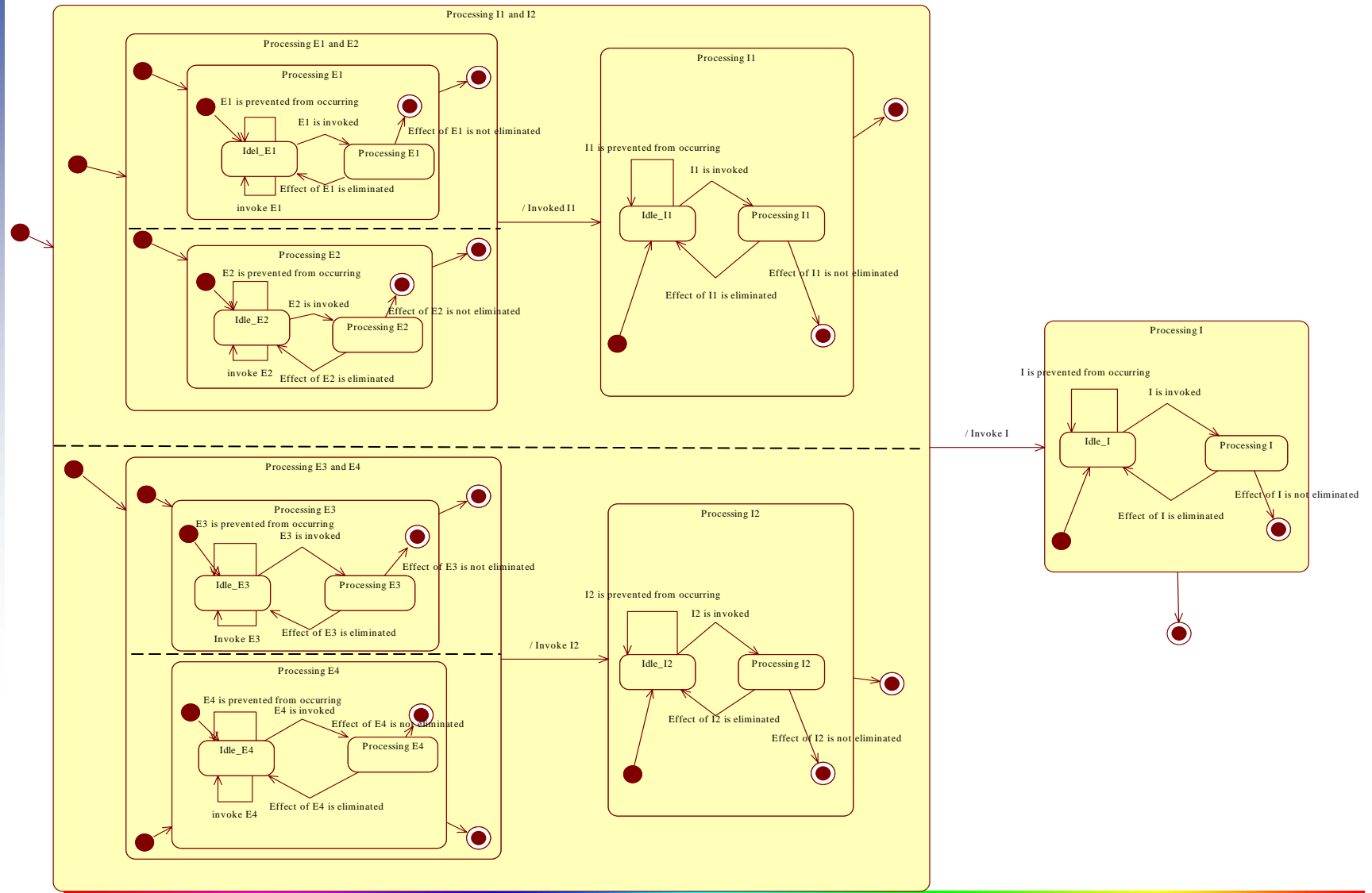


Minimal Cut Set Transition: Example 1

- A subtree for a given minimal cut set \mathcal{M} can be transformed to a minimal cut set transition
- The Boolean equations for the following subtree are $I = I_1 \cdot I_2$; $I_1 = E_1 \cdot E_2$; $I_2 = E_3 \cdot E_4$

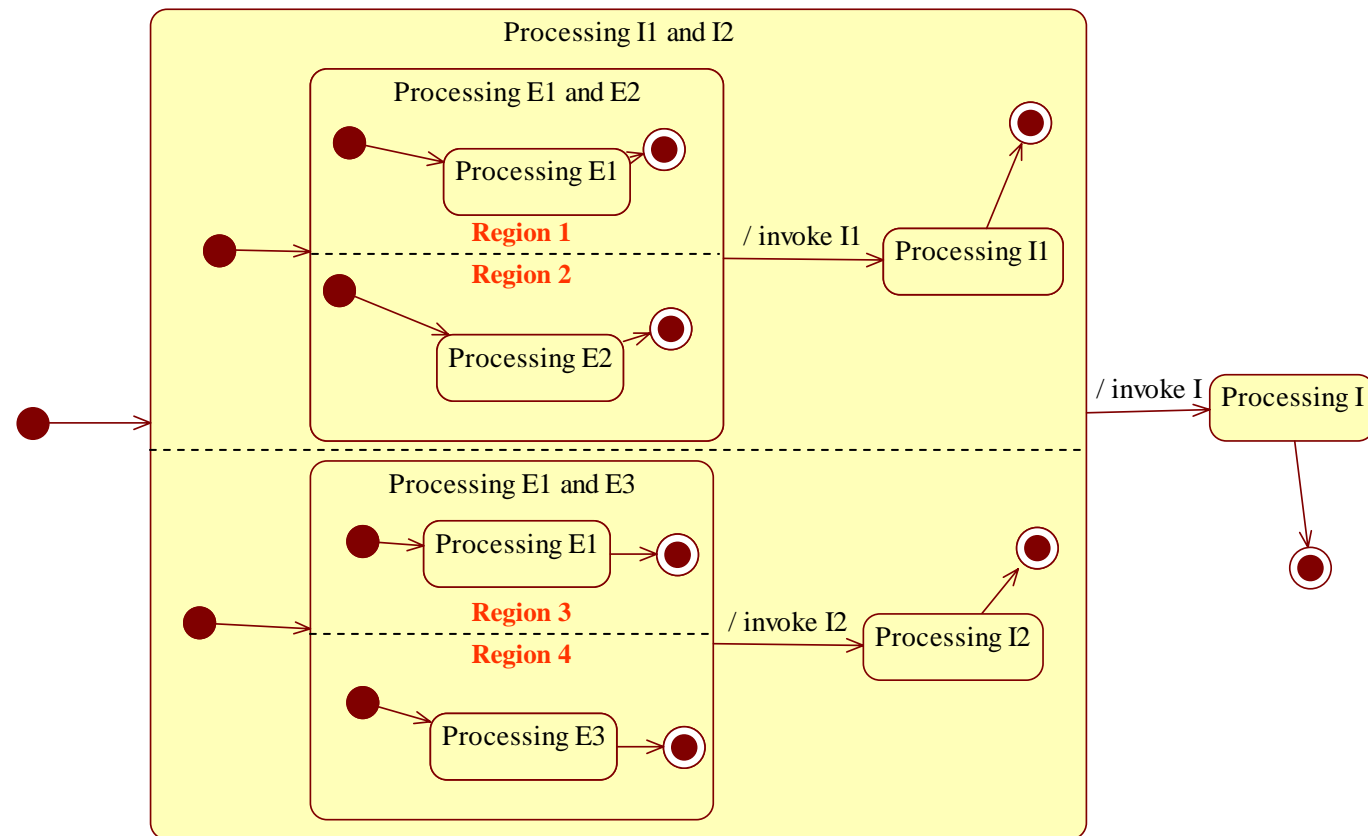


Minimal Cut Set Transition: Example 1 (cont'd)



Minimal Cut Set Transition: Example 2

- The Boolean equations for the following subtree are $I = I_1 \cdot I_2$; $I_1 = E_1 \cdot E_2$; $I_2 = E_1 \cdot E_3$



Event E_1 appears twice – once in Region 1 and once in Region 3

[Back to Research Issues](#)



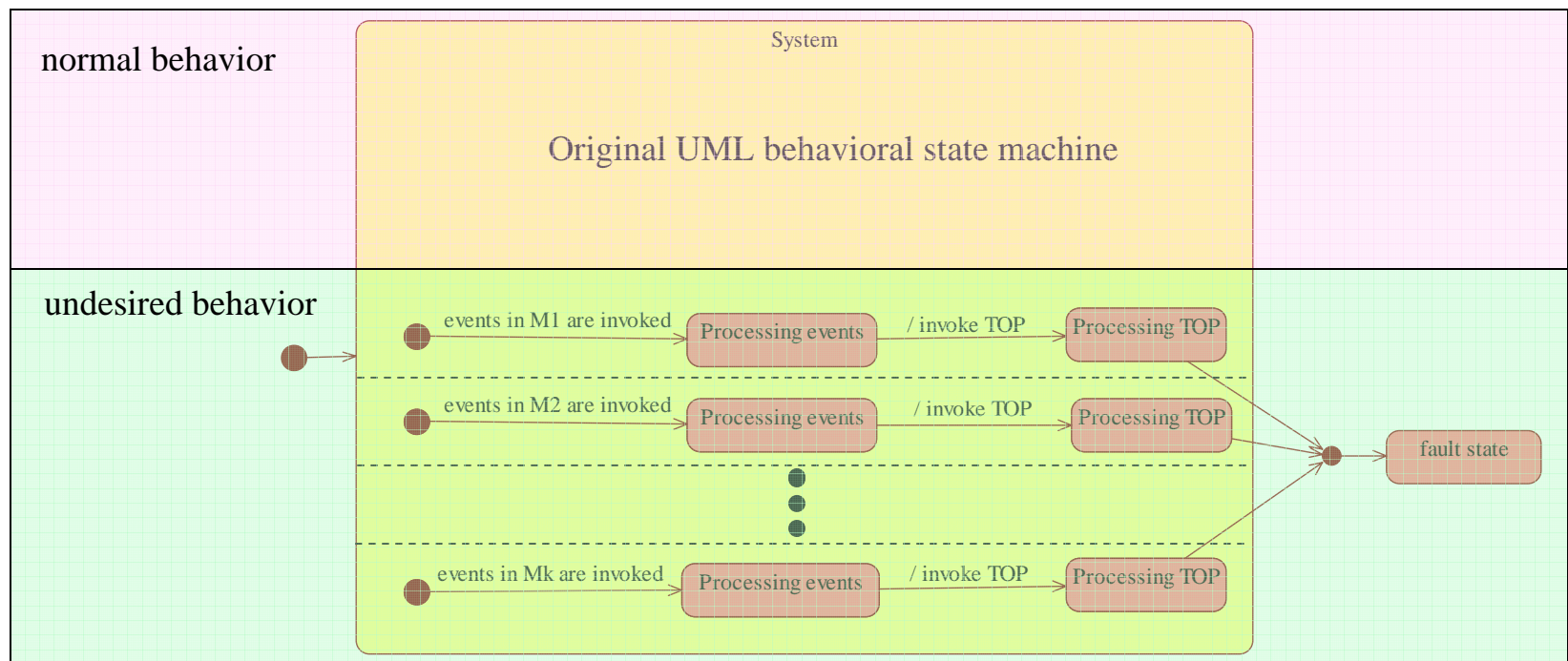
Research Issue 4

Add minimal cut set transitions to
a UML behavior state machine

[Back to Research Issues](#)

Combination of UML Machines & Fault Trees (1)

- Suppose a system has *one* undesired failure represented by the top event of a corresponding fault tree
- Suppose the top event $TOP = \mathcal{M}_1 + \mathcal{M}_2 + \dots + \mathcal{M}_k$ where \mathcal{M}_i ($1 \leq i \leq k$) is a minimal cut set for the top event and \mathcal{S}_i is the corresponding subtree (as discussed before) for \mathcal{M}_i
- We combine the original UML behavioral state machine and the fault tree by adding fault regions to the original machine, each of which contains a minimal cut set transition to a fault state



Combination of UML Machines & Fault Trees (2)

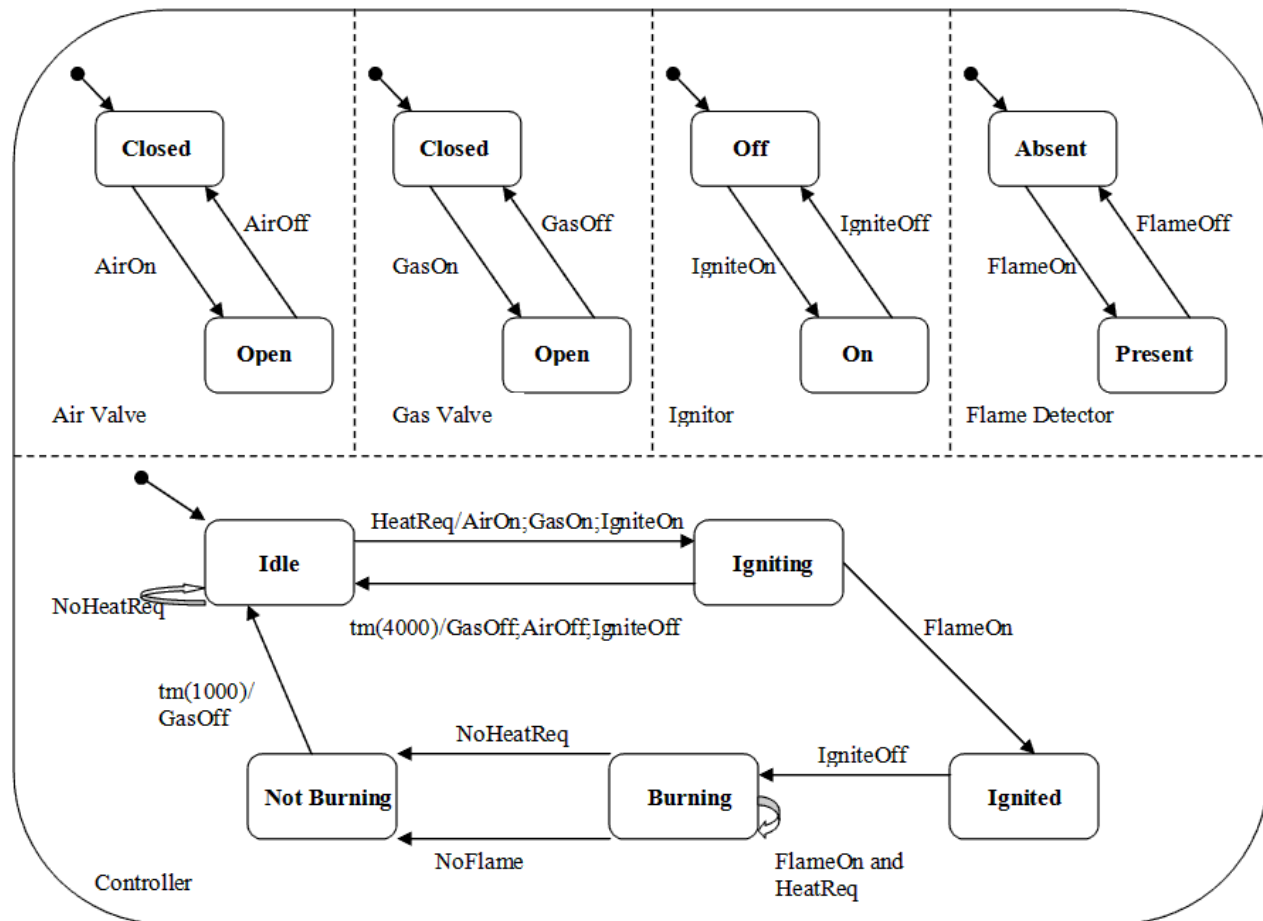
- All regions work in parallel. If any of the undesired regions reaches the fault state before the system terminates normally, then the system is not safe.
- If the system keeps running (i.e., never stops), then the system is not safe if the fault state can be reached from any of the fault regions.
- The same approach applies to a system with multiple undesired failures

A Gas Burner System (1)

- It consists of the following
 - an on/off valve to feed air
 - an on/off valve to feed fuel
 - a flame igniter
 - a flame detector
 - a controller
- The objectives of the *control system* for the burner are to
 - start it up
 - maintain it with an ignited flame
 - shut it down when requested
 - deal with abnormal and emergency conditions that may arise during operation

A Gas Burner System (2)

- Original UML behavioral state machine



A Gas Burner System (3)

- Boolean equations for the fault tree

$$TOP = I_1 + E_5; I_1 = E_1 \cdot I_2 \cdot E_4; I_2 = E_2 \cdot E_3$$

- Minimal cut set $\mathcal{M}_1 = \{E_5\}$

- Events in subtree $S_1 = \mathcal{M}_1 \cup \{TOP\}$
- Boolean equations for S_1 : $T = E_5$

- Minimal cut set $\mathcal{M}_2 = \{E_2, E_3\}$

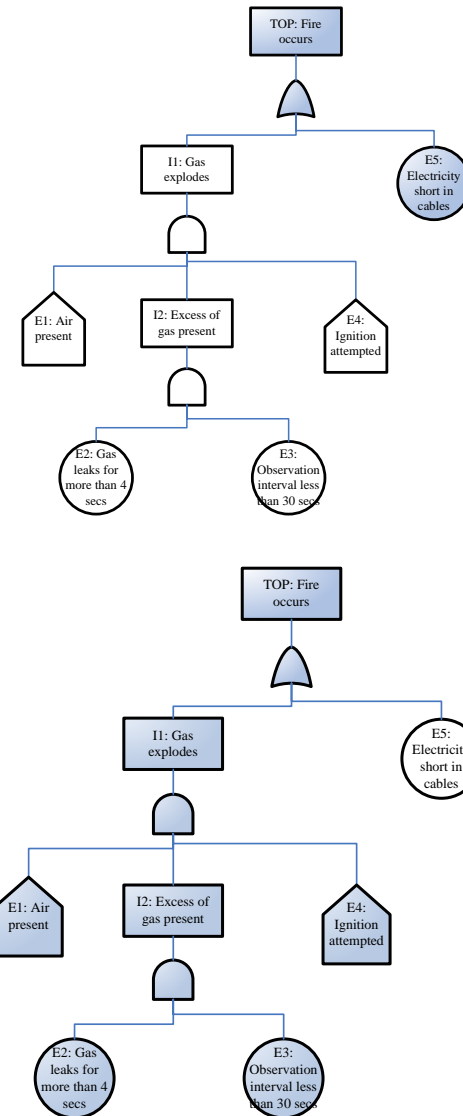
- Events in subtree $S_2 = \mathcal{M}_2 \cup \{E_1, E_4\} \cup \{I_1, I_2\} \cup \{TOP\}$

- Boolean equations for S_2 :

$$T = I_1$$

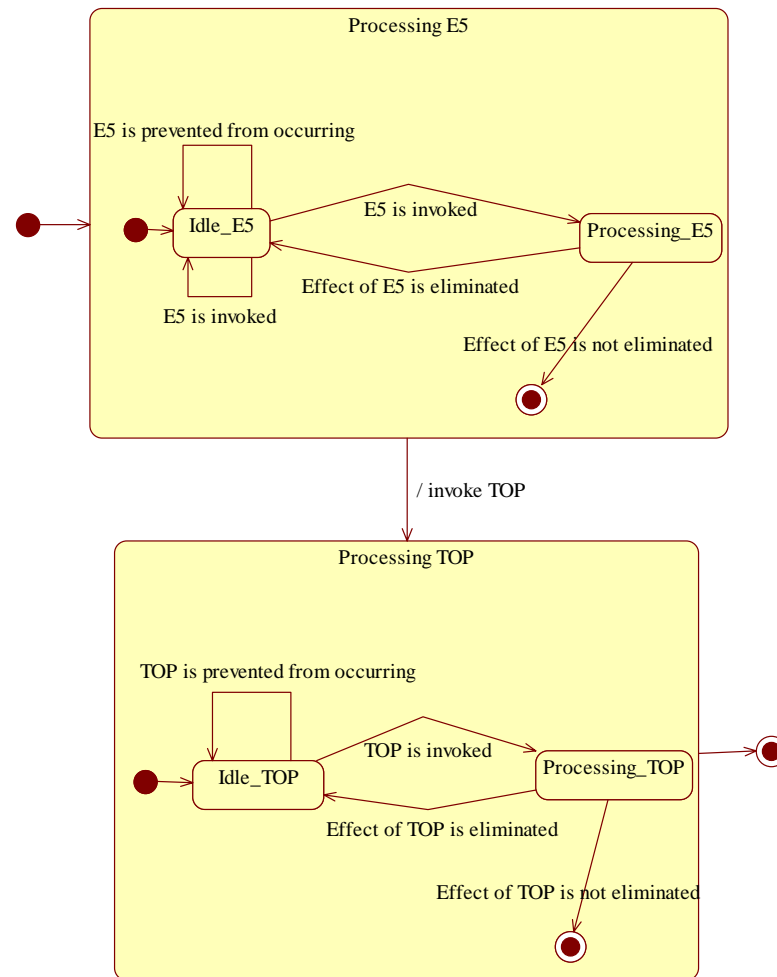
$$I_1 = E_1 \cdot I_2 \cdot E_4$$

$$I_2 = E_2 \cdot E_3$$



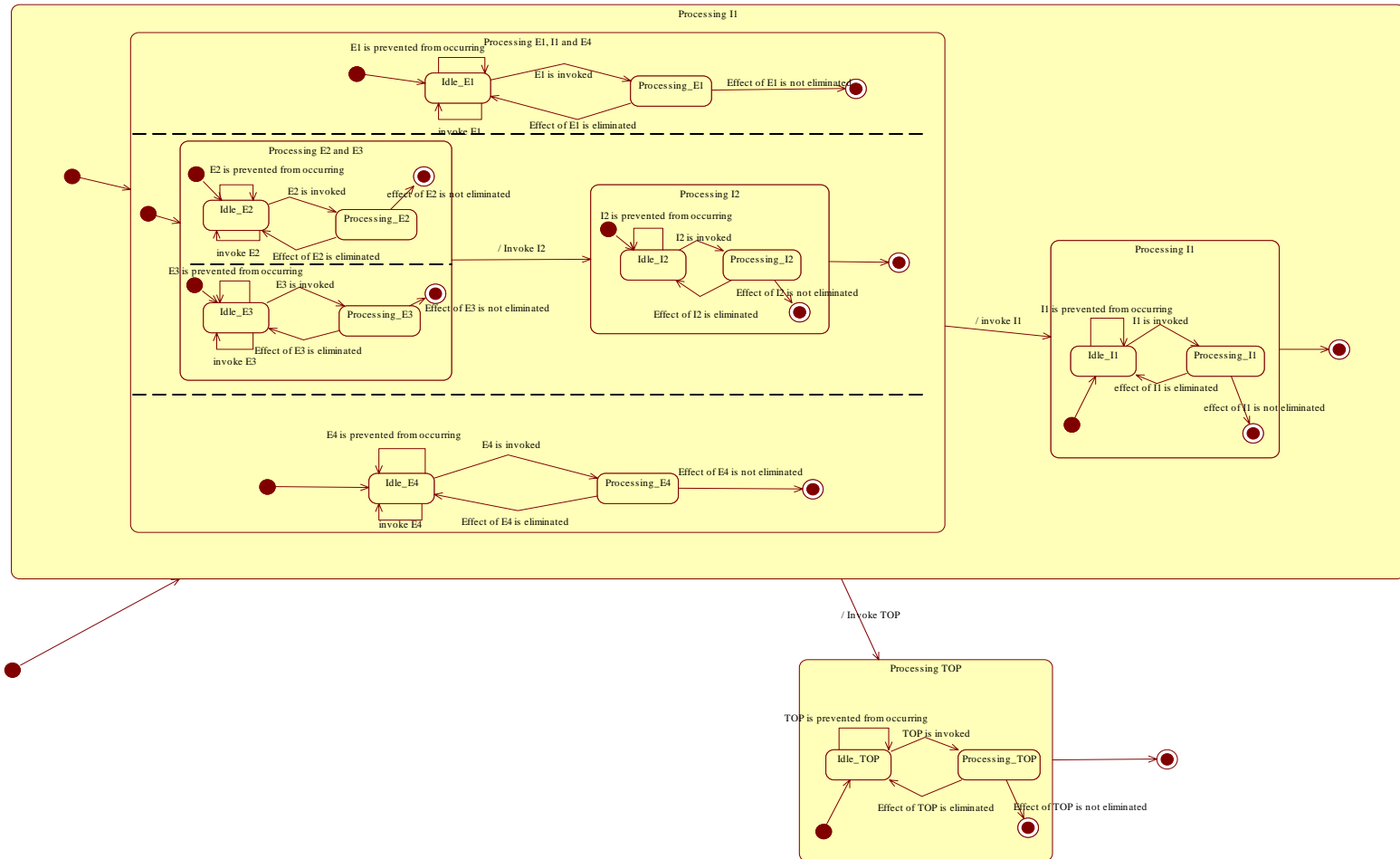
A Gas Burner System (4)

- Transform subtree \mathcal{S}_1 (for \mathcal{M}_1) to its corresponding minimal cut set transition



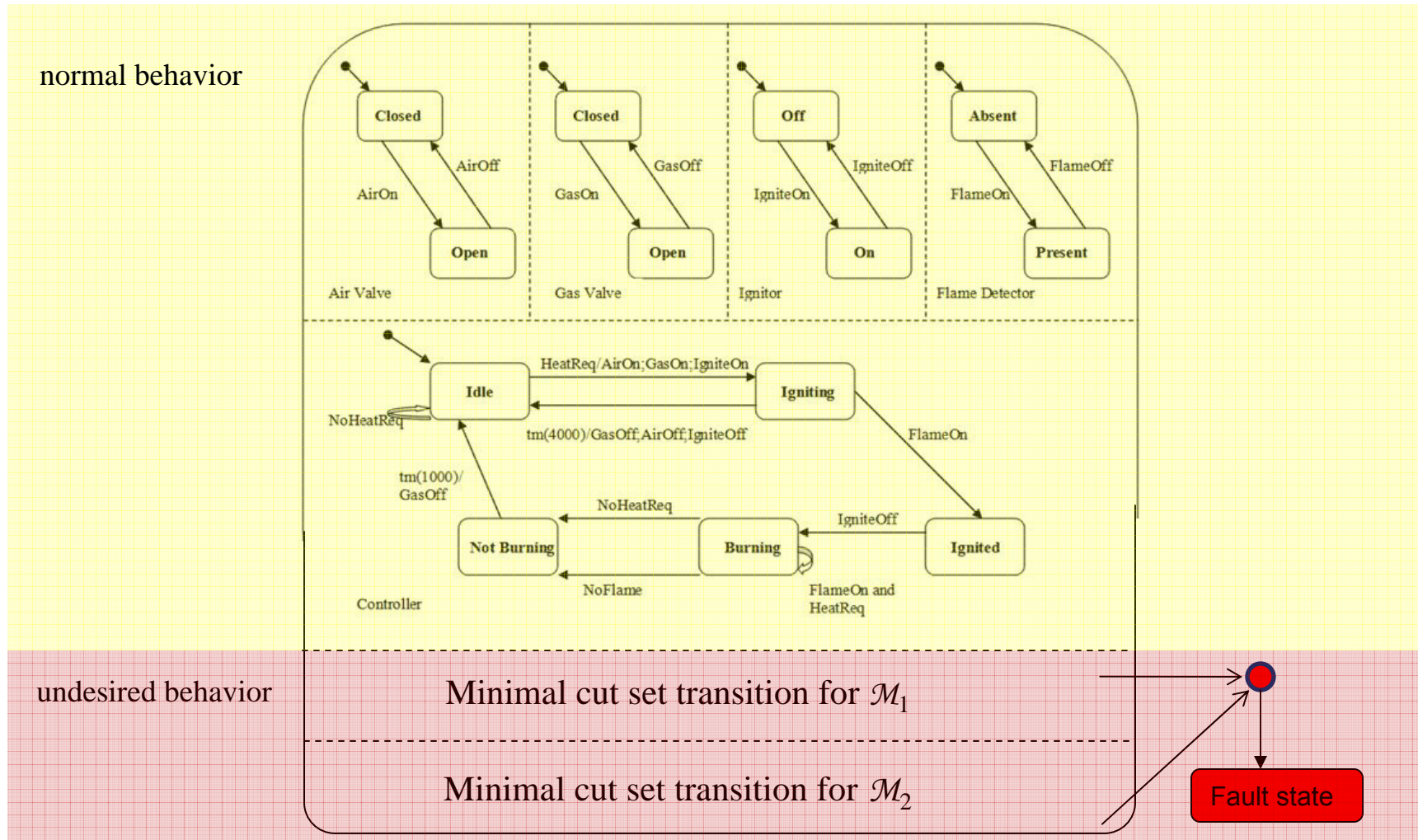
A Gas Burner System (5)

- Transform subtree S_2 (for \mathcal{M}_2) to its corresponding minimal cut set transition



A Gas Burner System (6)

- Add feasible minimal cut set transitions to the original behavioral state machine.





Conclusion

- Successfully developed a solution to integrate functional specifications in UML behavior state machines and hazard analysis in fault trees
- Writing a comprehensive report due on 12/31/2007